Revisiting Evolutionary Programming

David B. Fogel[1] and Kumar Chellapilla[2]

[1]Natural Selection. Inc., 3333 N. Torrey Pines Ct., Suite 200, La Jolla, CA 92037.
[2]University of California at San Diego, Dept. of ECE, La Jolla, CA 92093-4007.

ABSTRACT

Evolutionary programming is a method for simulating evolution that has been investigated for almost 40 years. When originally introduced, the available computing equipment was quite slow and difficult to use as measured by current standards. This paper provides a series of experiments that follow the framework of the original approach from the early 1960s, brought up to date with current computing machinery. A brief review of evolutionary programming and its relationship to other methods of evolutionary computation, specifically genetic algorithms and evolution strategies, is also offered.

Keywords: evolutionary programming, evolutionary computation, forecasting, control.

1. INTRODUCTION

There are three main lines of investigation within the current framework of evolutionary computation: (1) genetic algorithms, (2) evolution strategies, and (3) evolutionary programming. Reviews of these methods are offered in several recent books[1-5]. Each of these methods has developed over more than 30 years, and mostly independently of each other. Only recently have there been overt efforts to unify these very similar approaches to modeling the fundamental aspects of natural evolution.

In evolutionary programming, attention is placed on variation operations that can be constructed for a given representation so as to usefully adjust the behavior of the solutions in light of the performance measure for the task at hand. No attempt is made to model the overt mechanisms of genetics as found in nature simply for their own sake. Instead, general mathematical transformations are applied to solutions. There is no reason to believe that any particular mechanism of achieving a specified functionality will be productive in a simulation simply because it is observed in the natural world (e.g., modern aircraft are not ornithopters); the explicit and direct use of mechanisms such as allelic recombination, jumping genes, deletions, and so forth, may or may not be of any particular utility. Experience indicates that practitioners of evolutionary algorithms can often design more productive means for varying the solutions in a particular circumstance than is afforded by overtly mimicking natural genetic mechanisms[6-10].

This paper provides results from experiments that follow from the earliest efforts in evolutionary programming. A population of finite state machines is evolved to either (1) predict a sequence of temporal symbols from a generating "plant" with respect to a specified payoff function, or (2) use the best predictive machine at each point in time to attempt to control the plant that generates the observed sequence of symbols. When originally proposed (1960), the available computing machinery allowed for only very small population sizes and relatively slow computational speed. The current experiments seek to update this prior work in light of contemporary technology. A brief review of early evolutionary programming efforts is given before detailing the method and associated results obtained here.

2. EARLY EVOLUTIONARY PROGRAMMING

Evolutionary programming was devised by Lawrence J. Fogel in 1960 while serving at the National Science Foundation (NSF). Fogel was on leave from Convair, tasked as special assistant to the associate director (research), Dr. Richard Bolt, to study and write a report on investing in basic research. With regard to artificial intelligence, at the time it was concentrated mainly around heuristics and the simulation of primitive neural networks. It was clear to L. Fogel that both these approaches

---

1. E-mail: dfogel@natural-selection.com; WWW: http://www.natural-selection.com/people/dbf.html;
Tel: 619-455-6449; Fax: 619-455-1560.
2. E-mail: kumarc@ece.ucsd.edu; WWW: http://vision.ucsd.edu/~kchellap/;
Tel: 619-453-6748; Fax: 619-534-1004.

were limited because they model humans rather than the essential process that produces creatures of increasing intellect: evolution. L. Fogel considered intelligence to be based on adapting behavior to meet goals in a range of environments. In turn, prediction was viewed as the key ingredient to intelligent behavior and this suggested experiments on the use of simulated evolution of finite state machines to forecast nonstationary time series with respect to arbitrary criteria. These and other experiments were documented in a series of publications[11-14].

Intelligent behavior was viewed as requiring the composite ability to (1) predict one's environment, coupled with (2) a translation of the predictions into a suitable response in light of the given goal. For the sake of generality, the environment was described as a sequence of symbols taken from a finite alphabet. The evolutionary problem was defined as evolving an algorithm (essentially a program) that would operate on the sequence of symbols thus far observed in such a manner so as to produce an output symbol that is likely to maximize the algorithm's performance in light of both the next symbol to appear in the environment and a well-defined payoff function. Finite state machines (FSMs) provided a useful representation for the required behavior.

The proposal was as follows: A population of finite state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the payoff function (e.g., all-none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g., average payoff per symbol) indicates the fitness of the machine.

Offspring machines are created by randomly mutating each parent machine. Each parent produces offspring (this was originally implemented as only a single offspring per parent simply for convenience). There are five possible modes of random mutation that result naturally from the description of the machine: (1) change an output symbol, (2) change a state transition, (3) add a state, (4) delete a state, or (5) change the initial state. The deletion of a state and change of the initial state are only allowed when the parent machine has more than one state. Mutations are chosen with respect to a probability distribution, which is typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution or may be fixed *a priori*. These offspring are then evaluated over the existing environment in the same manner as their parents. Other mutations, such as majority logic operating on three or more machines, were proposed in Fogel *et al.*[14] but not implemented.

The machines that provide the greatest payoff (i.e., offer the most suitable predictions) are retained to become parents of the next generation. (Typically, half the total machines were saved so that the parent population remains at a constant size.) This process is iterated until an actual prediction of the next symbol (as yet unexperienced) in the environment is required. The best machine generates this prediction, the new symbol is added to the experienced environment, and the process is repeated. Fogel[12,14] used "nonregressive" evolution. To be retained, a machine had to rank in the best half of the population. Saving lesser-adapted machines was discussed as a possibility[14] but not incorporated.

This general procedure was successfully applied to problems in prediction, identification, and automatic control[14-16] and was extended to simulate coevolving populations[17]. Additional experiments evolving finite state machines for sequence prediction, pattern recognition, and gaming can also be found in several publications[18-22], many masters theses from New Mexico State University in the 1970s (see ref. 23), and others.

L. Fogel's original experiments were conducted on an IBM 704. This afforded the possibility for populations of three to five finite state machines, with a running speed of about one-half that of an Apple II or Commodore 64 (documented from multiple sources of computational speed comparisons of various machines over the development of the modern computer). This slow execution limited the possible sample size of trials, and therefore the possible statistical description of the behavior of the algorithm. It is of interest to revisit this original procedure in light of knowledge gained about evolutionary computation over the last 35+ years (e.g., with the inclusion of self-adaptation and tournament selection) and assess some of the procedure's statistical properties concerning prediction and control using current technology.

3. METHOD

Evolutionary programming (EP) was used to evolve a population of FSMs for prediction and control as follows:

1. *Initialization*: The initial population consisted of μ machines, $\boldsymbol{P}_i = (N_i, S_i, \boldsymbol{L}_i, \boldsymbol{O}_i, \lambda_i)\ \forall\ i \in \{1, 2, \ldots, \mu\}$, generated at random. The number of states in the machines, N_i, was selected at random uniformly from $\{1, 2, \ldots, N_{max}\}$, the

start state, S_i, and links, L_i, were assigned at random based on the number of states in the random machine, and the output symbols, O_i, were randomly assigned based on the number of symbols in the alphabet. Associated with each machine was one strategy parameter, λ_i, which represented the mean number of mutations to be applied to the parent to generate an offspring. The mean, λ_i, was initialized to 5. The generation count, k, was initialized to 1.

2. *Fitness Evaluation*: The fitness (or error) of the machine depended on the objective function for the task at hand, i.e., prediction or control, as will be described below.

3. *Mutation*: Each of the parents, P_i, was mutated to produce one offspring, P'_i, through the application of a sequence of mutation operators. The number of mutation operators, M_i, to be applied was a preset constant in the prediction experiments. In the control experiments, M_i was obtained by sampling a Poisson random variable with offspring's mean parameter, λ_i', obtained from the parent's mean parameter, λ_i, using

$$\lambda'_i = \lambda_i + 0.5N(0,1) \tag{1}$$

where $N(0,1)$ is a Gaussian random variable with mean zero and variance one. If λ_i' or M_i exceeded N_i or fell below 1, they were reset to the limit they violated. The mutation operators consisted of *adding a state*, *deleting a state*, *reassigning the start state*, *reassigning a link*, and *reassigning an output symbol*. If the parent already had N_{max} states then *adding a state* was precluded. Similarly, if the parent had only one state then *delete a state* and *reassign a start state* were precluded. The mutation operators are described below:

a) *Adding a state*: A new state was added to the parent machine. The new state's links and output symbols were randomly assigned. A random link from a pre-existing state in the parent machine was broken and linked to the newly created node, making it a potentially active part of the machine.

b) *Delete a state*: One of the existing states in the parent machine was randomly selected for deletion. All the links pointing to the selected state were reassigned randomly to other states. The selected state was deleted. If the selected state was the start state, then a new start state was chosen randomly.

c) *Reassign the start state*: A new start state was chosen at random.

d) *Reassign a link*: A randomly selected link was reassigned to a random different state.

e) *Reassign an output symbol*: The output symbol corresponding to a randomly chosen link was reassigned to a different symbol from the alphabet.

These five mutation operators were capable of generating a variety of genotypic variations in the parent machine. In most cases, adding, deleting, and reassigning the start state brought about a considerable change in behavior (defined as response to a stimulus sequence) whereas reassigning a link or an output symbol produced relatively minor changes.

4. *Fitness Evaluation*: Each offspring was scored in light of an evaluation function. The evaluation functions were different for the prediction and control experiments (see below).

5. *Selection*: For population sizes of less than 10, the population was sorted by fitness and the better half of the population was chosen to be the parents for the next generation. Otherwise, tournament selection was used to determine the parents. Pairwise comparisons were conducted over the union of parents, $\{P_i\}\ \forall\ i \in \{1, 2, \ldots, \mu\}$, and offspring $\{P'_i\}\ \forall\ i \in \{1, 2, \ldots, \mu\}$. Following current probabilistic selection in evolutionary programming[2], for each machine, q opponents were chosen randomly from all parents and offspring with equal probability. For each comparison, if the machine's predictive error was no greater than the opponent's, it received a 'win'. The μ individuals with the most wins were selected to be the parents for the next generation.

6. The procedure was terminated if the halting criterion was satisfied; otherwise, the generation number was incremented, i.e., $k = k + 1$, and the process proceeded to step 3. For the prediction experiments, the halting criterion was 5 generations, i.e., $k = 5$.

3.1 Prediction experiments

In the prediction experiments, the goal was to evolve a population of FSMs that would predict the next symbol of the periodic sequence consisting of (101110011101)* in one period[14,24-26]. Thus the alphabet for the FSMs in the prediction experiments was {0,1}. Evolution was started with a training set consisting of the first 20 symbols of the cyclic sequence, which served as the initial observation. A population of random FSMs was generated and evolved for five generations. The evaluation function

for evolving the population was the mean absolute error over the symbols in the training set. After this brief period of evolution, the best member in the population was used to predict the next symbol of the sequence. The prediction error was noted. This new symbol was added to the training set (as an additional observation) and the population was evolved for another five generations. This was repeated 300 times, resulting in a total of 1500 generations. This EP procedure was evaluated at each prediction using the cumulative fraction of correct predictions.

The prediction experiments were carried out with population sizes $\mu = \{3, 5, 10, 50, 100\}$, and the number of mutations, M_i, was chosen from $\{1, 2, 3\}$. Thirty trials were conducted for each combination of μ and M_i. The number of states in the target plant varied from 1 to 10. In all the experiments, the maximum number of states in the trial FSM, N_{max}, was set to 10.

3.2 Control experiments

In the control experiments, the goal was to evolve a population of FSMs that would determine the control input to a target plant (an FSM) that would cause it to output the (arbitrary) symbol 5. The members in the population represented models of the target plant that were "inverted" to obtain the control input symbol. The alphabet for the FSMs consisted of the numbers 0, 1, ... , 9.

In the first set of experiments, the target plant was an *N*-state *cyclic increment loopback FSM*, and in the second set of experiments, it was a random *N*-state FSM. A three-state cyclic increment loopback FSM is shown in Figure 1. The start state

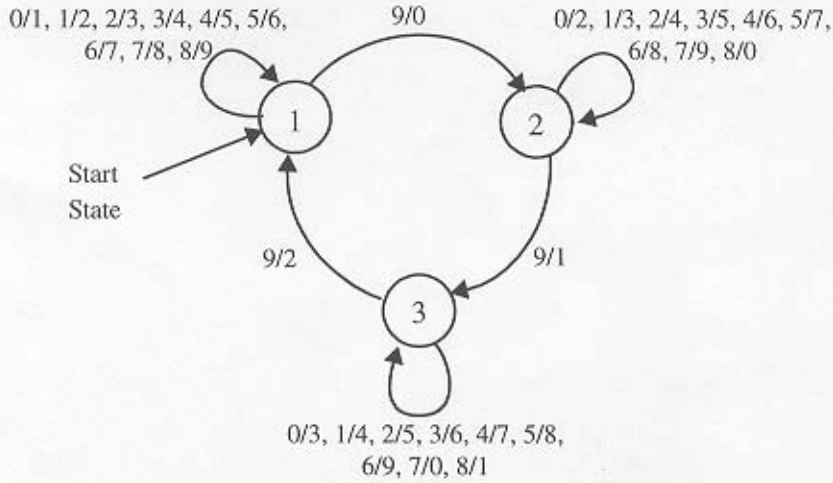

Figure 1. A three-state *cyclic increment loopback FSM*

was the first state, i.e., the state numbered 1 in the figure. When the current internal state of the cyclic increment loopback FSM was 1, input symbols 0, 1, ... , 8, generated outputs symbols 1, 2, ... , 8, 9, respectively. When a 9 was input, the output symbol was 0, and a state transition occurred from the first to the second state. Similarly, for the second state, input symbols 0, 1, ... , 8, generated output symbols 2, 3, ... , 9, 0, respectively. When a 9 was input, the output symbol was 1 and a state transition occurred from the second to the third state. In the last state, i.e., the third state in the machine, inputs 0, 1, ... , 8 produced outputs 3, 4, ... , 0, 1, respectively. For an input of 9, the output symbol was 2 and a state transition occurred from the third to the first state. An *N*-state cyclic increment loopback FSM can be defined using a similar logic. Notice that the *N*-state cyclic increment loopback FSMs were always controllable, i.e., at each instant it was possible to find an input symbol that would cause the FSM to output a 5. On the other hand, a random machine with *N* states might only be partially controllable (i.e., in certain states it might not possible to output a 5 for any input symbol), or uncontrollable (i.e., none of the states in the machine ever output a 5 for any input).

In the first set of control experiments, evolution began with a population of random machines. The number of states in each machine was randomly selected from $\{1, 2, ... , 10\}$. In the second set of experiments, while evolving controllers for *N*-state random machines, the initial population consisted of randomly generated FSMs with exactly *N* states and the *adding a state* and *deleting a state* mutations were not considered. An initial random sequence of N_{init} (see below) input symbols was given as input to the target plant and the output was observed. The resulting N_{init} pairs of input-output symbols were used as a training set to evolve the population of FSMs (using the above described EP procedure) for 100 generations. The evaluation function for this initial training was the sum absolute mismatch error over the N_{init} input-output pairs. After this initial training

phase a control step was performed. The best machine in the population was given all the observed inputs and was driven by this sequence into a final state, where it was "inverted" to obtain the control input that would cause it to output a 5. The "inversion" step consisted of examining all possible input-output pairs of the current state of the best machine and choosing an input symbol that caused the FSM to output a 5. In case a 5 could not be generated, a control input that produced the minimal possible deviation from 5 was selected. At any time, if more than a single choice for the input symbol was available, one was selected at random. The above computed control input was considered the best estimate of the input symbol that would cause the plant to output a 5. This control input became the next actual input to the plant and the corresponding output was observed. This new observation was added to the training set of observed input-output pairs and the population was evolved for 5 or 20 generations (depending on the size of the plant to be controlled, see below). This process of performing a control step, adding the new observation to the training set, and evolving the population was repeated N_C times. N_C was 200, 400, 600, 800, 1000, and 1000 for control experiments with 1, 2, 3, 4, 5, and 10-state FSMs, respectively, in both sets of experiments.

At each control step, the mean absolute deviation of the plant's output from 5 (over all past control steps) was used as a measure of the performance of the EP procedure in controlling the plant. Experiments were conducted using population sizes of 3, 5, 10, 50, and 100. The number of mutation operators, M_i, was determined using the mean parameter λ'_i, which was self-adapted as described in Eq. 1. Thirty trials were conducted for each population setting for each of the control settings. While selecting random N-state machines as candidate plants to be controlled, no attention was paid to ensure that they were controllable.

4. RESULTS

4.1 Prediction results

Figures 2(a)-(c) show the mean fraction of correct predictions of the best machine in the population averaged over 30 trials for population sizes of 3, 5, 10, 50, and 100, with the number of mutation operators equal to 1, 2, and 3, respectively. The first prediction was correct approximately 80 percent of the time (over 30 trials). The fraction of correct predictions then dropped, typically during a transient phase, followed by asymptotic climb towards an improved level of performance. Figure 3 shows the mean fraction of correct predictions (averaged over 30 runs) at the 300th generation for different population sizes as a function of the number of mutations used to generate offspring. No statistically significant differences were observed between trials with the same population size that used different numbers of mutations to generate offspring. In light of space limitations, results obtained when using a single mutation are discussed. Table 1 provides the mean fraction of correct predictions (averaged over 30 runs) at the end of 300 predictions. With a population size of 3, only 2 of the 30 trials produced FSMs that could predict all 12 symbols correctly. The FSMs in the remaining 28 trials could predict 11 of the 12 symbols correctly. When the population size was increased to 100, 15 of the 30 trials produced perfect solutions, and the remaining 15 produced FSMs that could correctly predict 11 symbols. Table 2 gives the results from the t-test for statistical significance[27] for differences in the cumulative fraction of correct predictions.

4.2 Control results

Figure 4 shows a typical cumulative mean absolute deviation curve when EP was used to evolve a population of 50 FSMs to control a one-state cyclic increment loopback FSM. A perfect controller was found at generation 4, but this was randomly lost in subsequent generations. For brief periods starting at generations 13, 25, 29, 37, and 71, the perfect controller was replaced by imperfect solutions that overfit the observed data. However, a perfect solution was quickly regained and the curve asymptotically tended towards zero error. A similar curve for a five-state cyclic increment loopback FSM is given in Fig. 5. A perfect controller was first found at generation 123. Occasionally the population overfit the observed data and the best member in the population made errors, which were observed at generations 186 and 350. But these were again quickly corrected. Table 3 summarizes the results from the first set of control experiments with the cyclic increment loopback FSMs.

Figure 6 shows a typical cumulative mean absolute deviation curve while controlling a random five-state machine with a population of FSMs evolved using the EP procedure. The random machine was completely controllable and a perfect solution was found at the 168th control step (the curve asymptotically tended towards zero absolute deviation). Figure 7 shows a similar curve while controlling a random five-state machine that was only partially controllable. The closest output symbol to 5 that the machine could generate was a 6. An optimal controller that could control the plant to output a 6 was found at the 182nd control step and the curve asymptotically tended to the mean best possible deviation of 1.

Of the 30 trials with the one-state random plant, 20 trials contained fully controllable plants, 7 contained partially control-

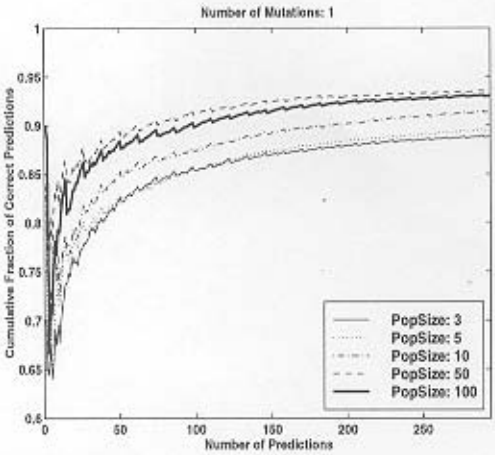

Figure 2(a). The cumulative fraction correct of the best machine in the population at each prediction averaged over 30 trials with the EP method applied to the environment (101110011101)*. A single mutation operator was used.

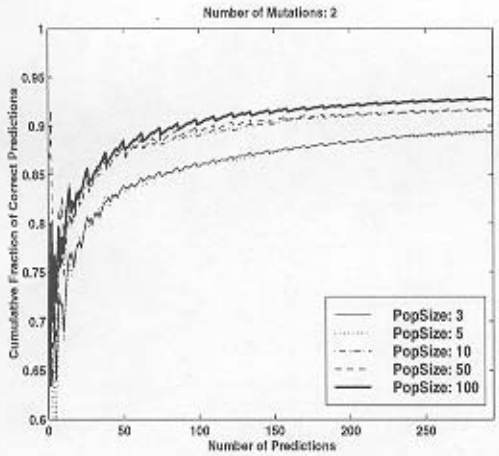

Figure 2(b). The cumulative fraction correct of the best machine in the population at each prediction averaged over 30 trials with the EP method applied to the environment (101110011101)*. Two mutation operators were used.

lable plants that could be controlled with an asymptotic mean absolute deviation of 1, and the remaining 3 plants were also partially controllable but with an asymptotic mean absolute deviation of 2. In all 30 trials, the EP procedure was successful in finding the best possible controllers within 200 control steps. In the experiments with two-state random machines, in 28 of the 30 trials, EP was successful in finding optimal controllers. Of these, 15 were fully controllable, 3 were partially controllable with asymptotic errors of 0.5, 7 were partially controllable with asymptotic errors of 1.0, the remaining 3 were partially controllable with asymptotic errors of 1.5, 2.0, and 3.0. Similarly, for the 3-, 4-, 5-, and 10-state machines the number of successful runs (with asymptotic convergence to the best available performance) were 24, 17, 12, and 4, respectively. Table 4 summarizes the results from the control experiments with random plants.

5. DISCUSSION

Although the original efforts of Fogel *et al.*[14], and others, indicated success in evolving finite state machines for predicting time series and controlling the output of other finite state machines, the available computing hardware did not allow for a statistical design that would demonstrate robust performance. The current experiments indicate this evidence. A trend toward more rapid convergence to better predictive machines was generated with an increased population size, but reliable performance was generated even with small population sizes (as in Fogel *et al.*[14]). Moreover, the evolutionary programming procedure was able to efficiently generate optimal controllers for random finite state machines of up to 10 states with 10 input-output symbols[†]. The evidence suggests that the primary efforts in evolutionary programming were decades ahead of the computer equipment required to validate their utility.

Evolutionary programming was originally proposed as a procedure for generating machine intelligence. Intelligence was viewed in the context of adaptive behavior, predicting the environment and taking suitable action in light of a given purpose. The same basic procedure can be extended and used for numerical optimization in pattern discovery, system identification, automatic control, and multiplayer gaming (control of an intelligent adversary). The representation for each problem follows from the task at hand; there is no specific requirement for using any particular coding (e.g., a finite state machine or a bit string). Similarly, rather than prescribe the use of any particular transformation operator (e.g., crossover), variations are applied to parent solutions using operators that follow naturally from the coding structure. If the codings are real-valued then Gaussian mutations (or Cauchy) often provide for an efficient search. If a number of elements must be altered, a Poisson random variable with an appropriately selected (or self-adaptive) rate may provide a suitable operation. The crucial element in

†. It may be of interest to note that the size of the search space in a problem with 10 states and 10 input and output symbols is 10^{200}. Thus the evidence here (as in Fogel *et al.*[14]) contradicts the claim that evolutionary programming was "insufficiently powerful to search other than small problem spaces quickly"[30] (p. 106).

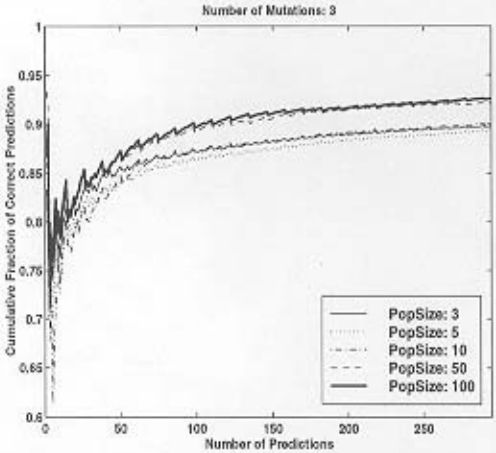

Figure 2(c). The cumulative fraction correct of the best machine in the population at each prediction averaged over 30 trials with the EP method applied to the environment (101110011101)*. Three mutation operators were used.

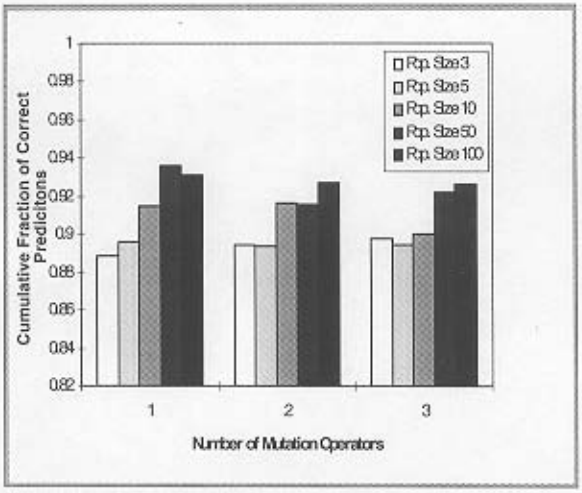

Figure 3. The cumulative fraction correct of the best machine in the population at the 300th prediction averaged over 30 trials with the EP method applied to the environment (101110011101)*. For a given population size, the differences corresponding to a change in the number of mutations used to generate offspring were not statistically significant.

constructing an appropriate mutation operation is that it must maintain a suitable behavioral link between each parent and its offspring.

The initial efforts of L.J. Fogel indicate some of the early attempts to (1) use simulated evolution to perform prediction, (2) include variable-length encodings, (3) use representations that take the form of a sequence of instructions, (4) incorporate a population of candidate solutions, and (5) to coevolve evolutionary programs. Moreover, Fogel[12,14,28] offered the early recognition that natural evolution and the human endeavor of the scientific method are essentially similar processes, a notion echoed later in Gell-Mann[29]. The initial prescriptions for operating on finite state machines have been extended to arbitrary representations, mutation operators, and selection methods, and techniques for self-adapting the evolutionary search have been proposed and implemented. The population size need not be kept constant and there can be a variable number of offspring per parent, much like the (μ+λ) methods offered in evolution strategies. In contrast to these methods, selection is often made probabilistic in evolutionary programming giving lesser-scoring solutions some probability of surviving as parents into the next generation. In contrast to genetic algorithms, no effort is made in evolutionary programming to maximize schemata processing, nor is the use of random variation constrained to emphasize specific mechanisms of genetic transfer, perhaps providing greater versatility to tackle specific problem domains that are unsuitable for genetic operators such as crossover.

Although evolutionary programming, evolution strategies, and genetic algorithms have some notable differences, their similarities are in many ways more important. Each of these general procedures, developed over the last 30-35+ years, has demonstrated the utility of stochastic optimization using a population-based search. As increased attention is focused on these optimization techniques, there will be greater synergy not only between alternative methods of evolutionary computation, but also between other research communities such as operations research, numerical analysis, optimization, and artificial intelligence. The authors hope that the innovative roots of evolutionary computation can be reconsidered in light of our current abilities and knowledge.

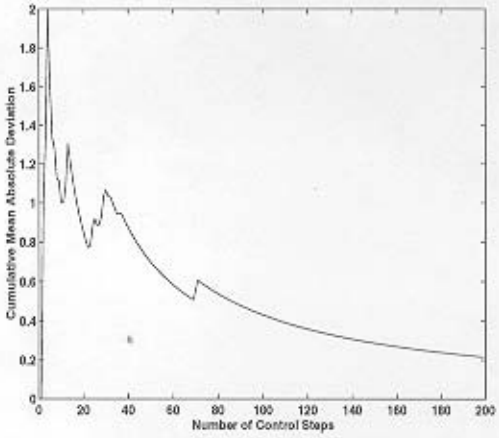

Figure 4. A typical cumulative mean absolute deviation curve as a function of the number of control steps while evolving a population of 50 FSMs to control a one-state *cyclic increment loopback FSM*. A perfect solution was first found at generation 4. At generations 13, 25, 29, 37, and 71, it was replaced by partially perfect solutions that overfit the available observed input-output sequence. However, a perfect solution was quickly regained.

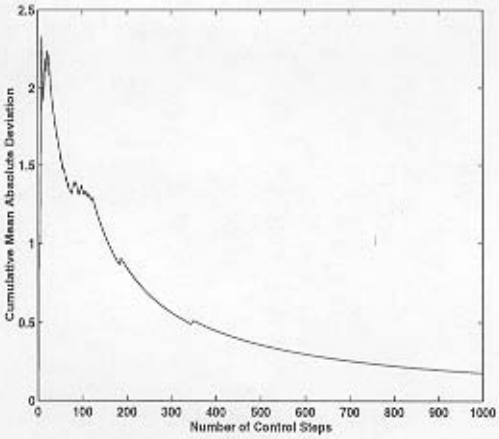

Figure 5. A typical cumulative mean absolute deviation curve as a function of the number of control steps while evolving a population of 50 FSMs to control a five-state *cyclic increment loopback FSM*. A perfect solution was first found at generation 123. Occasionally the population overfit the past observed data and the best member in the population made control errors, which can be observed at generations 186 and 350. But these were quickly corrected.

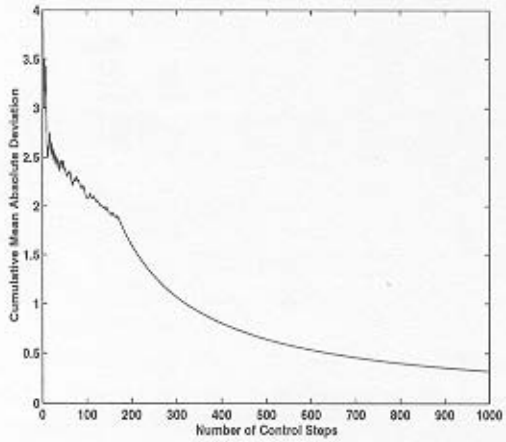

Figure 6. A typical cumulative mean absolute deviation curve as a function of the number of control steps while evolving a population of 50 FSMs to control a "controllable" random five-state FSM. A perfect solution was found at the 168th control step.

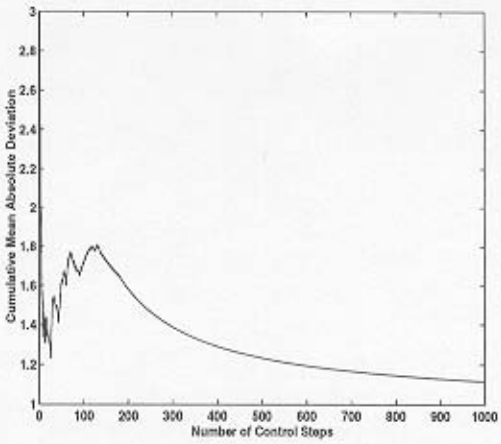

Figure 7. A typical cumulative mean absolute deviation curve as a function of the number of control steps while evolving a population of 50 FSMs to control a "partially controllable" random five-state FSM. An optimal controller was found at the 182nd control step, having an aymptotic mean absolute deviation of 1.

Population Size	Mean Fraction Correct (at 300th prediction)	Standard Deviation
3	0.8888	0.03252
5	0.8961	0.03646
10	0.9149	0.03933
50	0.9358	0.04514
100	0.9310	0.04389

Table 1: The mean fraction of correct predictions (over 30 runs) at the end of 300 predictions. A single mutation was used to generate offspring.

		Population Size				
		3	5	10	50	100
Population Size	3	0	-0.80	-2.77*	-4.58*	-4.42*
	5	0.80	0	-1.82	-3.83*	-3.31*
	10	2.77*	1.82	0	-1.87	-1.63
	50	4.58*	3.83*	1.87	0	0.40
	100	4.42*	3.31*	1.63	-0.40	0

Table 2: *T*-test results for comparisons between the cumulative fraction of correct predictions when using the EP method with different population sizes. Positive values indicate that the population size corresponding to the row is better. The results that generated *P*-values < 0.05 are statistically significant and are indicated by an asterisk (*).

Number of states (N)	Number of control steps (N_C)	Training set size for the initial evolution step (N_{init})	Number of generations of evolution between successive control steps	Number of successful runs (out of 30)	Cumulative mean absolute deviation (over all past N_C control steps) averaged over all 30 runs
1	200	20	5	21	0.9245
2	400	20	5	25	0.7038
3	600	50	20	27	0.4485
4	800	100	20	28	0.3287
5	1000	100	20	21	0.5536
10	1000	100	20	11	1.9348

Table 3: Results from the first set of control experiments on the *cyclic increment loopback FSMs*. A run was considered successful if a perfect controller that caused the target plant to consistently output a 5 was found. Once a perfect controller is found the cumulative mean absolute deviation starts to asymptotically approach zero error.

Number of states (N)	Number of control steps (N_C)	Training set size for the initial evolution step lasting 100 generations N_{init}	Number of generations of evolution between two successive control steps	Number of successful runs (out of 30)*	Cumulative mean absolute deviation (over all past N_C control steps) averaged over the successful runs
1	200	20	5	20 (0), 7 (1), 3 (2)	0.4643
2	400	20	5	15 (0), 3 (0.5), 7 (1), 1 (1.5), 1 (2), 1 (3),	0.6404
3	600	100	20	24	0.8592
4	800	200	20	17	1.3666
5	1000	1000	20	12	1.6763
10	1000	1000	20	4	2.4681

Table 4: Results from the second set of control experiments with randomly generated machines. (*)Along with the number of successful runs, where available, the numbers in parentheses indicate the asymptotic cumulative mean absolute deviation.

6. REFERENCES

1. H.-P. Schwefel, *Evolution and Optimum Seeking*, John Wiley & Sons, NY, 1995.
2. D.B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, IEEE Press, NY, 1995.
3. T. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford, NY, 1996.
4. M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, Cambridge, MA, 1996.
5. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd ed., Springer, Berlin, 1996.
6. D.B. Fogel and J.W. Atmar, "Comparing Genetic Operators with Gaussian Mutations in Simulated Evolutionary Processes Using Linear Systems," *Biological Cybernetics*, vol. 63, pp. 111-114, 1990.
7. M.M. Rizki, L.A. Tamburino, and M.A. Zmuda, "Evolving Multi-Resolution Feature Detectors," *Proceedings of the Second Annual Conference on Evolutionary Programming*, D.B. Fogel and W. Atmar (eds.), Evolutionary Programming Society, La Jolla, CA, pp. 108-118, 1993.
8. D.B. Fogel and L.C. Stayton, "On the Effectiveness of Crossover in Simulated Evolutionary Optimization," *BioSystems*, Vol. 32:3, pp. 171-182, 1994.
9. D.J. Nettleton and R. Garigliano, "Evolutionary Algorithms and a Fractal Inverse Problem," *BioSystems*, Vol. 33:3, pp. 221-232, 1994.
10. D.K. Gehlhaar, G.M. Verkhivker, P.A. Rejto, C.J. Sherman, D.B. Fogel, L.J. Fogel, and S.T. Freer, "Molecular Recognition of the Inhibitor AG-1343 by HIV-1 Protease: Conformationally Flexible Docking by Evolutionary Programming," *Chemistry and Biology*, Vol 2:5, 317-324, 1995.
11. L.J. Fogel, "Autonomous Automata," *Industrial Research,* Vol. 4:2, pp. 14-19, 1962.
12. L.J. Fogel, "On the Organization of Intellect," Ph.D. Dissertation, UCLA, 1964.
13. L.J. Fogel, A.J. Owens, and M.J. Walsh, "Artificial Intelligence through a Simulation of Evolution," *Biophysics and Cybernetics Systems*, A. Callahan, M. Maxfield, and L.J. Fogel (eds.), Spartan Books, Washington DC, pp. 131-156, 1965.
14. L.J. Fogel, A.J. Owens, and M.J. Walsh, *Artificial Intelligence through Simulated Evolution*, John Wiley, NY, 1966.
15. L.J. Fogel, A.J. Owens, and M.J. Walsh, "On the Evolution of Artificial Intelligence," *Proc. of the 5th National Symp. on Human Factors in Electronics*, IEEE, San Diego, CA, pp. 63-76, 1964.
16. L.J. Fogel, "Extending Communication and Control through Simulated Evolution," *Bioengineering — An Engineering View: Proceedings of the Symp. Engineering Significance of the Biological Sciences*, G. Bugliarello (ed.), San Francisco Press, San Francisco, pp. 286-304, 1968.
17. L.J. Fogel and G.H. Burgin, "Competitive Goal-Seeking through Evolutionary Programming," Final Report under Contract no. AF19(628)-5927, Air Force Cambridge Research Labs, 1969.
18. B.E. Lutter and R.C. Huntsinger, "Engineering Applications of Finite Automata," *Simulation*, Vol. 13, pp. 5-11, 1969.
19. G.H. Burgin, "On Playing Two-Person Zero-Sum Games against Nonminimax Players," *IEEE Trans. Systems Science and Cybernetics*, Vol. SSC-5:4, pp. 369-370, 1969.
20. J.W. Atmar, "Speculation on the Evolution of Intelligence and Its Possible Realization in Machine Form," Doctoral Dissertation, New Mexico State University, Las Cruces, NM, 1976.
21. D.W. Dearholt, "Some Experiments on Generalization Using Evolving Automata," *Proceedings of the 9th Intern. Conf. on System Sciences*, Honolulu, pp. 131-13, 1976.
22. A. Takeuchi, "Evolutionary Automata — Comparison of Automaton Behavior and Restle's Learning Model," *Information Science,* Vol. 20, pp. 91-99, 1980.
23. D. B. Fogel (ed.) *Evolutionary Computation: The Fossil Record*, IEEE Press, Piscataway, NJ, 1998, forthcoming.
24. L.J. Fogel, P.J. Angeline, and D.B. Fogel, "An Evolutionary Programming Approach to Self-Adaptation on Finite State Machines," *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, MIT Press, Cambridge, MA, pp. 355-365, 1995.
25. L.J. Fogel, P.J. Angeline, and D. B. Fogel, "A Preliminary Investigation on Extending Evolutionary Programming to Include Self-adaptation on Finite State Machines," *Informatica* 18: 387-398, 1995.
26. P.J. Angeline, D.B. Fogel, and L.J. Fogel, "A Comparison of Self-Adaptation Methods for Finite State Machines in Dynamic Environments," *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, L.J. Fogel, P.J. Angeline, and T. Bäck (eds.), MIT Press, Cambridge, MA, pp. 441-449, 1996.
27. W.H. Press, S.A. Teukolsky, W.T. Vetterking, and B.P. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed., Cambridge University Press, NY, 1992.
28. L.J. Fogel, *Biotechnology: Concepts and Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1963.
29. M. Gell-Mann, *The Quark and the Jaguar*, Freeman Press, NY, 1994.
30. D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, MA: Addison-Wesley, 1989.